

Ans 1: let X_1, X_2, \dots, X_n be the random variables such that
 $X_i = 1$ If i^{th} person gets his umbrella.
 $X_i = 0$ otherwise.

Let total number of people who get their umbrella is $Y = \sum_{i=1}^n X_i$

$$E[Y] = E\left[\sum_{i=1}^n X_i\right] = \sum_{i=1}^n E[X_i]$$

Now, probability that the i^{th} person gets his umbrella is $P(X_i)$.
 $E[X_i] = P(X_i = 1) = (1/n)$.

$$E[Y] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n (1/n) = (1/n) \sum_{i=1}^n 1 = (1/n) * n = 1.$$

Ans 2: Let X_{ij} be the random variable such that

$X_{ij} = 1$ If (i,j) is an inverse pair.
 $X_{ij} = 0$ otherwise.

Also, a pair is equally likely to be an inversion or not. So $P(X_{ij} = 1) = 1/2$

Let total number of inversion pairs be $Y = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}$

$$\begin{aligned} E[Y] &= \sum_{i=1}^n \sum_{j=i+1}^{n-1} P[X_{ij} = 1] = \sum_{i=1}^n \sum_{j=i+1}^{n-1} (1/2) = (1/2) \sum_{i=1}^n (n-i) \\ &= (n^2 - n)/4 = n(n-1)/4 \end{aligned}$$

Ans 4:

a) --

```
Find_Element(A,p,r,k)
1. if( p >= r)
2.     return A[p]
3. q = partition(A,p,r)
4. if(q == k)
5.     return A[q]
6. if(q > k)
7.     return Find_Element(A,p,q- 1,k)
8. else
9.     return Find_Element(A,q+1,r,k)
```

```
Partition(A,p,r) {
1. n = A[r]
2. i = p- 1
3. for(j = p to r- 1)
    a. if(A[j] <= n)
    b.     i = i + 1
        exchange A[i] & A[j]
4. exchange A[i+1] & A[r]
5. return i+1
```

Worst case complexity- - - The worst case happens when the array is already sorted in non- decreasing order. For each call of partition , the last element of the array is chosen as a partition . At each level the time taken is $O(l)$ for the array of length l .

The recursion tree looks like :

$$T(n) = T(n- 1) + cn$$

$$T(n) = cn + c(n- 1) \dots + ck = c(n+k)(n- k+1)/2 = O(n^2)$$

Average case complexity –

In an average case, the call to partition will return the middle element of the array.

Therefore, $T(n) = T(n/2) + cn$

$$T(n) \leq c(n/2 + n/2^2 + \dots) = cn \left(\frac{1}{1 - (1/2)} \right) = 2cn$$

$$T(n) = O(n)$$

b) ---

```

Randomized_Find_Element(A,p,r,k)
1. if( p >= r)
2.     return A[p]
3. q = Randomized_partition(A,p,r)
4 if(q == k)
5     return A[q]
6 if(q > k)
7     return Randomized_Find_Element(A,p,q-1,k)
8 else
9     return Randomized_Find_Element(A,q+1,r,k)

```

```

Randomized_Partition(A,p,r) {
1. i = Random(p,r)
2. exchange A[i] & A[r].
3. return Partitio(A,p,r).

```

Average case complexity –

On an average case the call to the partition will return the middle element . Thus,

$T(n) = T(n/2) + cn + dn$; where dn is the time taken to calculate I in the first step of `Randomized_partition`.

$$T(n) = T(n/2) + C^1(n)$$

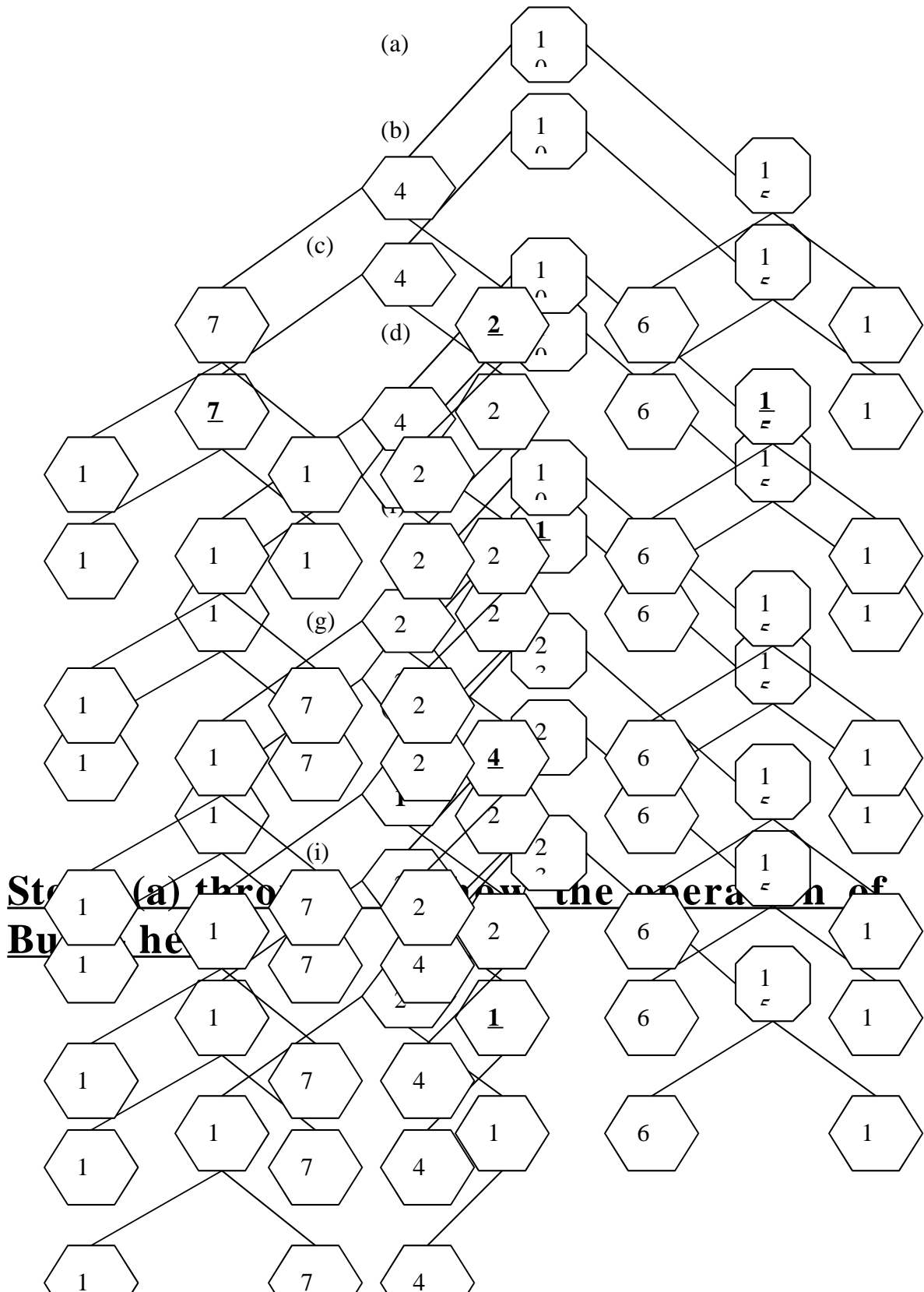
$$T(n) = C^1(n) + C^1(n/2) + C^1(n/2^2) + \dots$$

$$T(n) \leq n C^1 \left(\frac{1}{1 - (1/2)} \right) = 2n C^1$$

$$T(n) = O(n)$$

Ans 3:

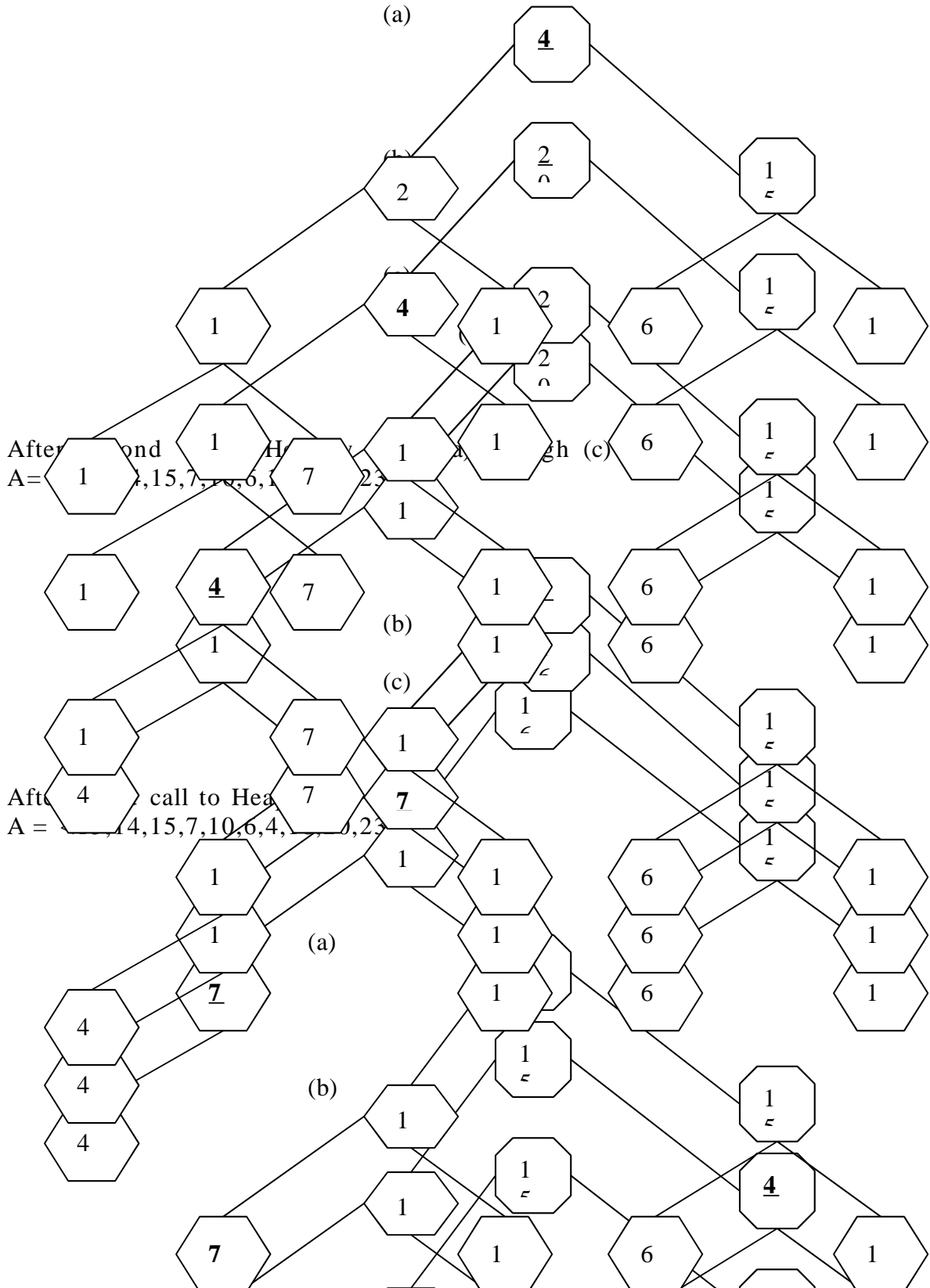
Build-heap(A)

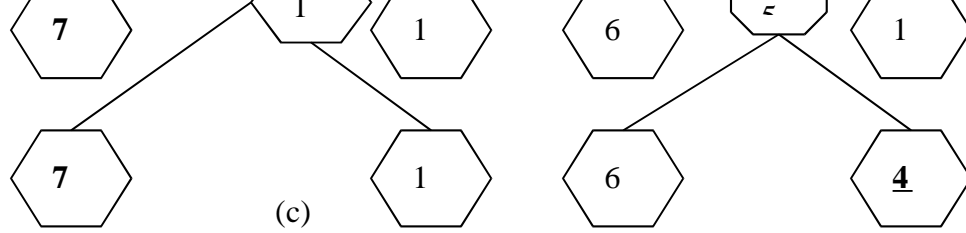


Heapify procedure- - - -

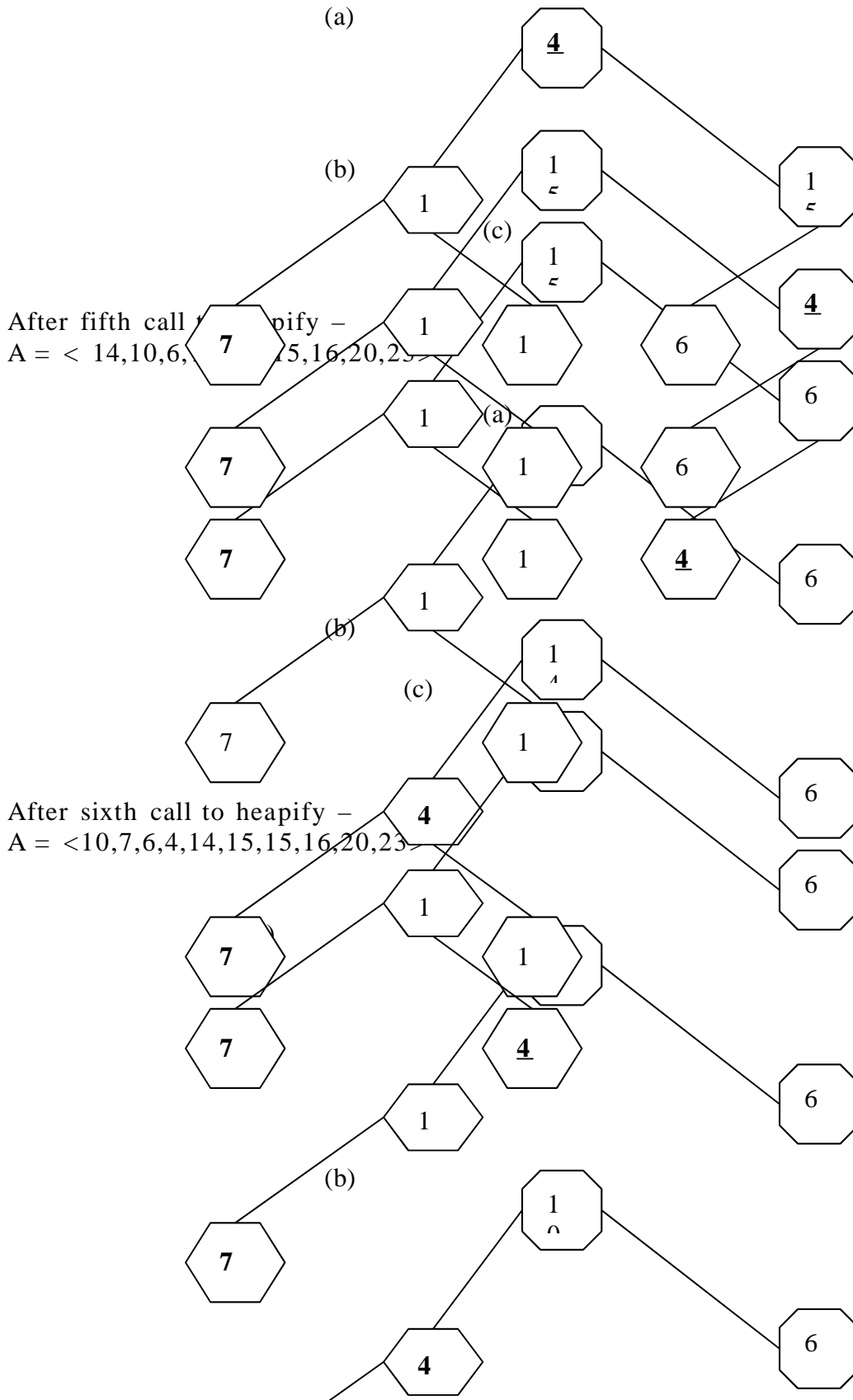
After First call to Heapify steps (a) through (d)

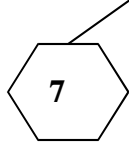
A = < 20,16,15,14,10,6,15,4,7,23 >



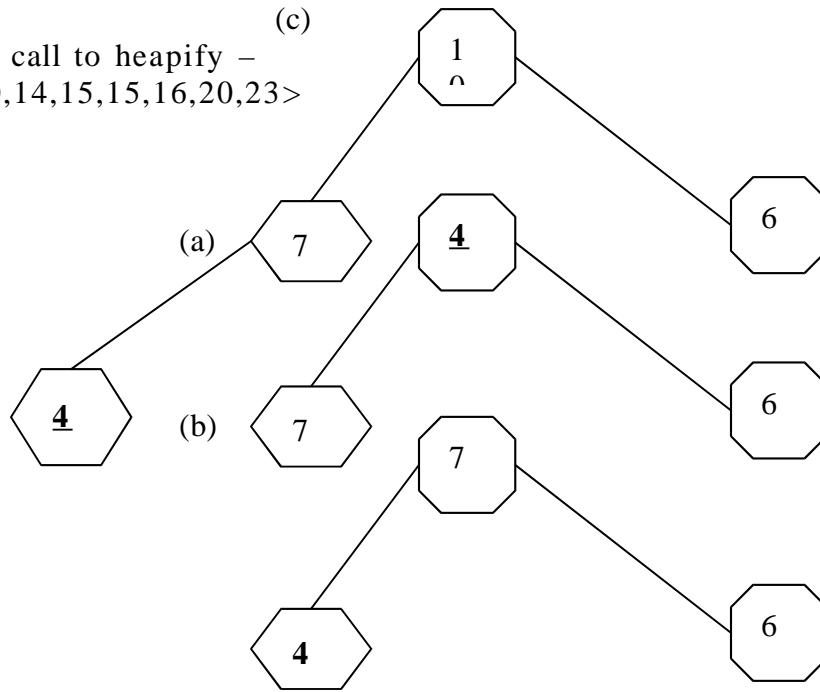


After fourth call to Heapify -
 A = <15,14,6,7,10,4,15,16,20,23 >

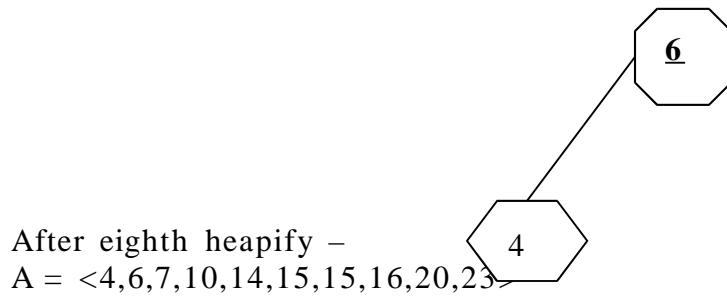




(c)
 After seventh call to heapify -
 A = <7,4,6,10,14,15,15,16,20,23>



After eighth call to Heapify - -
 A = <6,4,7,10,14,15,15,16,20,23>



After eighth heapify -
 A = <4,6,7,10,14,15,15,16,20,23>

